TDA - Thread Dump Analyzer

TDA - Thread Dump Analyzer

Published September, 2008

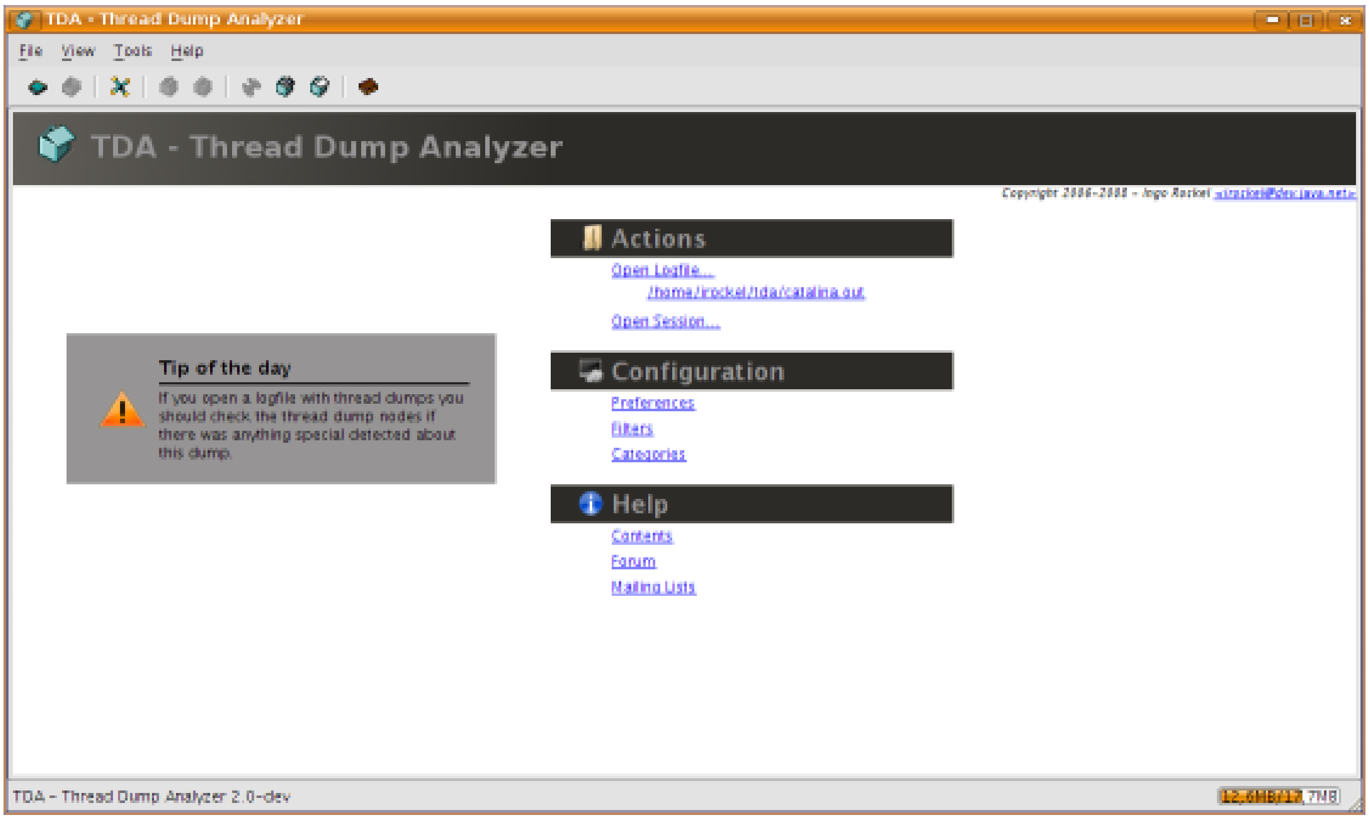
Copyright © 2006-2008 Ingo Rockel <irockel@dev.java.net>

Table of Contents

List of Figures

1
General

This chapter gives an introduction on how to use the TDA - Thread Dump Analyzer. TDA parses your log files and displays all found thread dumps and class histograms reported from a Sun JVM 1.4.x or better, SAP VM or HP-UX VM. Class Histograms are not included in the thread dumps by default but need a special JVM-Flag to be dumped with the thread dump (see below). As TDA does everything offline and thread dumps have very low impact on the VM (including the class histogram option), it can be used for production environments.


TDA - Thread Dump Analyzer
File View Tools Help
TDA - Thread Dump Analyzer
Copyright 2006-2008 - Ingo Rockel <irockel@dev.java.net>
Actions
Open Logfile...
/home/irockel/tda/catalina.out
Open Session...
Configuration
Preferences
Filters
Categories
Help
Contents
Forum
Mailing Lists
TDA - Thread Dump Analyzer 2.0-dev
12,6MB/17,7MB

Tip of the day

If you open a logfile with thread dumps you should check the thread dump nodes if there was anything special detected about this dump.

Figure 1.1. Welcome Screen

TDA can also be used as plugin in JConsole or VisualVM for requesting thread dumps from a remote virtual machine using JMX. See JConsole Plugin for further information on using it as JConsole Plugin and see VisualVM Plugin for information concerning VisualVM..

1.1. Request Thread Dumps

There are different methods to request a thread dump from a running applications, either using a kill Signal on UNIX, using a little tool on windows, since Java 1.5 using jstack or via Java Management Extensions (JMX).

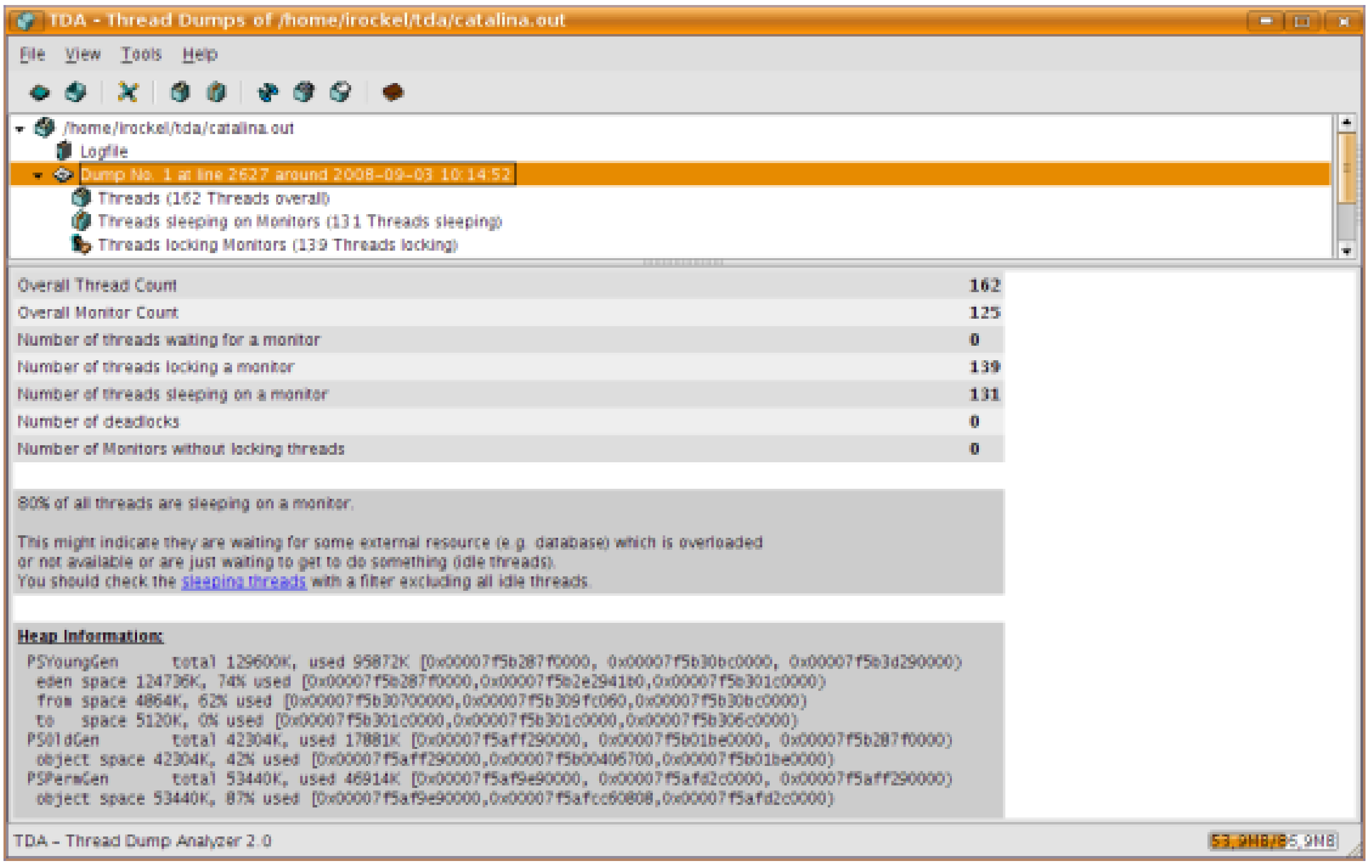
Using a kill-Signal. For requesting a thread dump, you need to send a kill-QUIT-Signal to the Java VM. For this you the PID of the VM is required and then do a `kill -QUIT <PID>` in a Unix Environment. On Windows there is a special tool for accomplishing this, e.g. SendSignal. See the References for a link to the homepage of this tool. It does the same like kill -QUIT on Unix. Note though, if you're using the remote connection on windows, you need to start the connection with `mstsc.exe /console` otherwise you won't have sufficient rights to use the tool. The JDK will write the thread-dump to standard-out. E.g. if your application is running in a tomcat, this would be catalina.out in a default tomcat installation. In other situations this might just be the console where the application was started from.

Using Java Management Extensions (JMX). Since Sun's Java Version 1.5 JMX is bundled with the JDK. The JDK also includes a small JMX client application *JConsole*, which can be used to connect to a running application using JMX. TDA is available as a plugin for JConsole. JMX is also available with VisualVM, a new troubleshooting tool. included in recent Java 1.6 JVM's. See Application Analysis using JConsole for detailed information on how to use JConsole with the TDA Plugin.

Using jstack. Also starting with Sun's JDK 1.5 you can also request a thread dump using the JVM utility jstack. jstack prints the thread dump into the shell window where you called it. To use the dump in TDA, pipe the output into a log file and open this file. To analyze multiple dumps just append them to the log file. jstack also allows you to get thread dumps from javacore-files, either dumped on an *OutOfMemoryError* or using the JVM tool jmap. On the windows platform this tool is available starting with JDK 1.6. It is not available at all on the linux-ia64 platform.

1.2. Thread Dump Parsing

Open the log file you want to analyse, TDA will search for all thread dumps in this log file and displays them in a tree. Up to Sun's JDK 1.5 the VM doesn't log any date information when the dump was requested, you can provide a regular expression in the *Preferences* which is then used for parsing the lines before the thread dump to get a time stamp (see below for an example). In JDK 1.6 a time stamp is logged just before the dump, the default regular expression is set up to parse this time stamp. In Regular Expression For Time Stamps you will find information about how to set up your own regular expression for time stamp parsing.


TDA - Thread Dumps of /home/irockel/tda/catalina.out
File View Tools Help
/home/irockel/tda/catalina.out
Logfile
Dump No. 1 at line 2627 around 2008-09-03 10:14:52
Threads (162 Threads overall)
Threads sleeping on Monitors (131 Threads sleeping)
Threads locking Monitors (139 Threads locking)

Overall Thread Count	**162**
Overall Monitor Count	**125**
Number of threads waiting for a monitor	**0**
Number of threads locking a monitor	**139**
Number of threads sleeping on a monitor	**131**
Number of deadlocks	**0**
Number of Monitors without locking threads	**0**

80% of all threads are sleeping on a monitor.

This might indicate they are waiting for some external resource (e.g. database) which is overloaded
or not available or are just waiting to get to do something (idle threads).
You should check the sleeping threads with a filter excluding all idle threads.

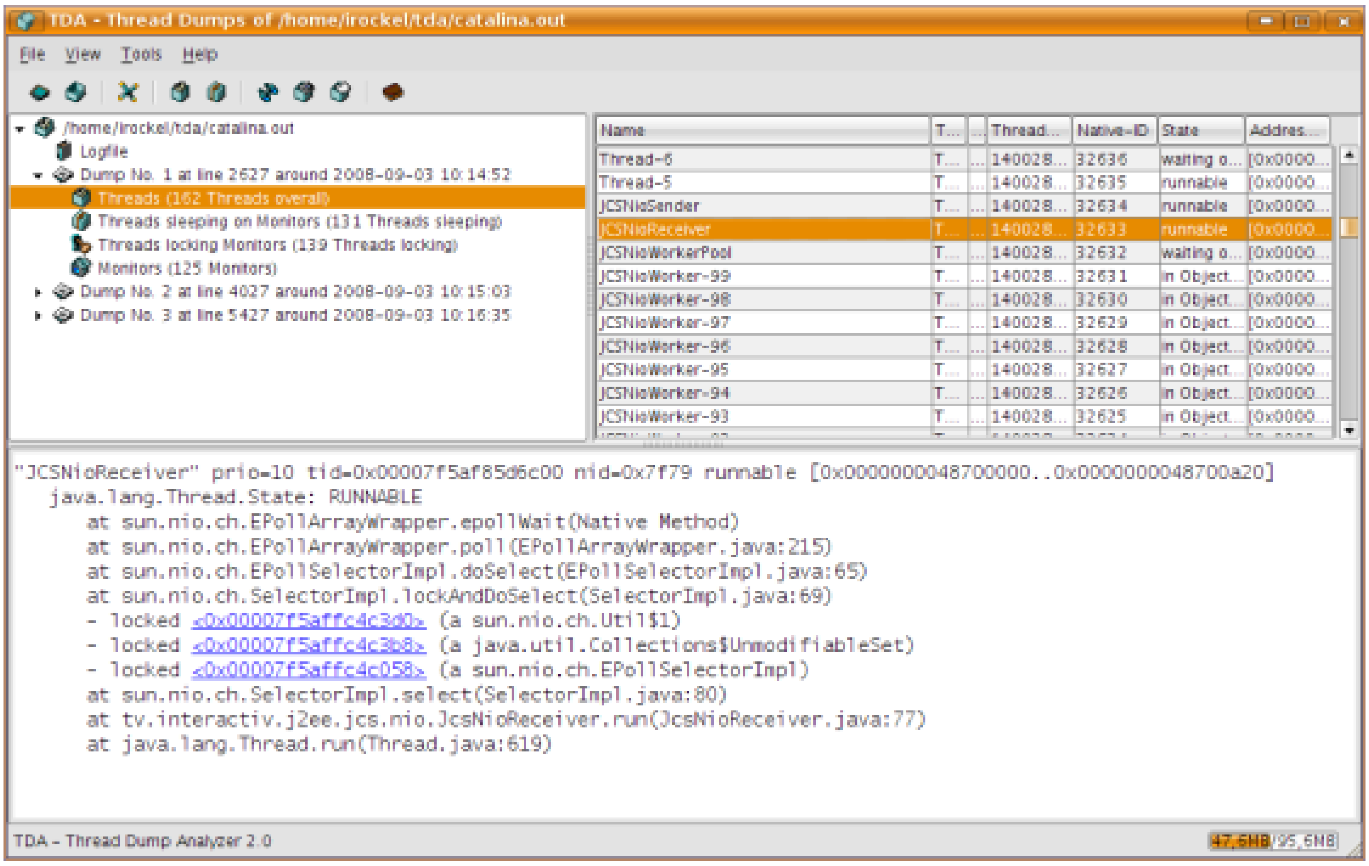
Heap Information:

```
PSYoungGen      total 129600K, used 95872K [0x00007f5b287f0000, 0x00007f5b30bc0000, 0x00007f5b3d290000)
 eden space 124736K, 74% used [0x00007f5b287f0000,0x00007f5b2e2941b0,0x00007f5b301c0000)
 from space 4864K, 62% used [0x00007f5b30700000,0x00007f5b309fc060,0x00007f5b30bc0000)
 to   space 5120K, 0% used [0x00007f5b301c0000,0x00007f5b301c0000,0x00007f5b306c0000)
PSOldGen        total 42304K, used 17881K [0x00007f5aff290000, 0x00007f5b01be0000, 0x00007f5b287f0000)
 object space 42304K, 42% used [0x00007f5aff290000,0x00007f5b00406700,0x00007f5b01be0000)
PSPermGen       total 53440K, used 46914K [0x00007f5af9e90000, 0x00007f5afd2c0000, 0x00007f5aff290000)
 object space 53440K, 87% used [0x00007f5af9e90000,0x00007f5afcc60808,0x00007f5afd2c0000)
```

TDA - Thread Dump Analyzer 2.0   53,9MB/85,9MB

Figure 1.2. Information about the selected dump

For each thread dump TDA sums up all found threads and all monitors found in the threads, it also groups threads waiting on, for and locking monitors. Clicking on the dump nodes itself will provide you with information about the dump. If there is something special about it, TDA will give you some hints about what it found and what to do next.



TDA - Thread Dumps of /home/irockel/tda/catalina.out

File View Tools Help

/home/irockel/tda/catalina.out
Logfile
Dump No. 1 at line 2627 around 2008-09-03 10:14:52
Threads (162 Threads overall)
Threads sleeping on Monitors (131 Threads sleeping)
Threads locking Monitors (139 Threads locking)
Monitors (125 Monitors)
Dump No. 2 at line 4027 around 2008-09-03 10:15:03
Dump No. 3 at line 5427 around 2008-09-03 10:16:35

Name	T...		Thread...	Native-ID	State	Addres...
Thread-6	T...	...	140028...	32636	waiting o...	[0x0000...
Thread-5	T...	...	140028...	32635	runnable	[0x0000...
JCSNioSender	T...	...	140028...	32634	runnable	[0x0000...
JCSNioReceiver	T...	...	140028...	32633	runnable	[0x0000...
JCSNioWorkerPool	T...	...	140028...	32632	waiting o...	[0x0000...
JCSNioWorker-99	T...	...	140028...	32631	in Object...	[0x0000...
JCSNioWorker-98	T...	...	140028...	32630	in Object...	[0x0000...
JCSNioWorker-97	T...	...	140028...	32629	in Object...	[0x0000...
JCSNioWorker-96	T...	...	140028...	32628	in Object...	[0x0000...
JCSNioWorker-95	T...	...	140028...	32627	in Object...	[0x0000...
JCSNioWorker-94	T...	...	140028...	32626	in Object...	[0x0000...
JCSNioWorker-93	T...	...	140028...	32625	in Object...	[0x0000...

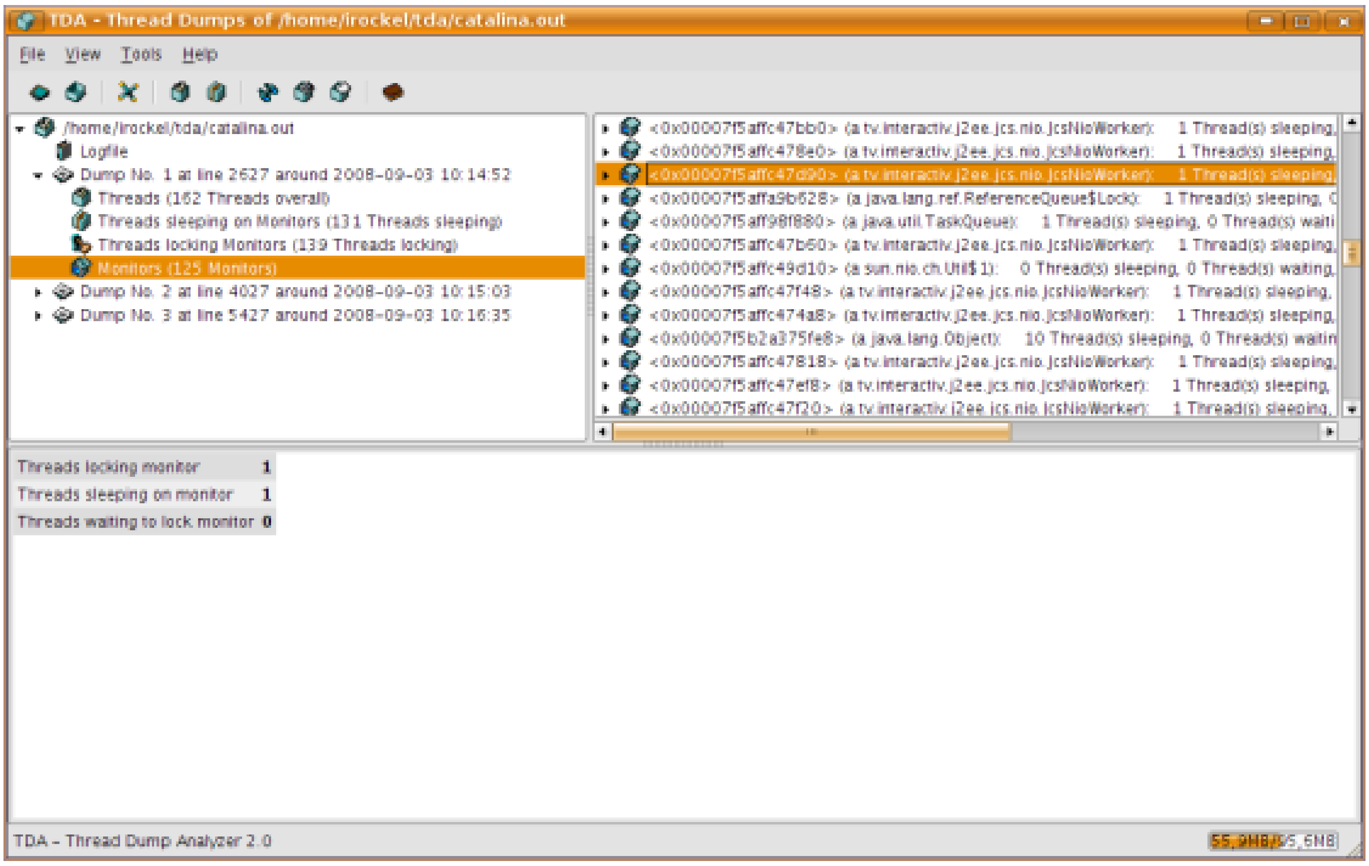
```
"JCSNioReceiver" prio=10 tid=0x00007f5af85d6c00 nid=0x7f79 runnable [0x0000000048700000..0x0000000048700a20]
   java.lang.Thread.State: RUNNABLE
      at sun.nio.ch.EPollArrayWrapper.epollWait(Native Method)
      at sun.nio.ch.EPollArrayWrapper.poll(EPollArrayWrapper.java:215)
      at sun.nio.ch.EPollSelectorImpl.doSelect(EPollSelectorImpl.java:65)
      at sun.nio.ch.SelectorImpl.lockAndDoSelect(SelectorImpl.java:69)
      - locked <0x00007f5affc4c3d0> (a sun.nio.ch.Util$1)
      - locked <0x00007f5affc4c3b8> (a java.util.Collections$UnmodifiableSet)
      - locked <0x00007f5affc4c058> (a sun.nio.ch.EPollSelectorImpl)
      at sun.nio.ch.SelectorImpl.select(SelectorImpl.java:80)
      at tv.interactiv.j2ee.jcs.nio.JcsNioReceiver.run(JcsNioReceiver.java:77)
      at java.lang.Thread.run(Thread.java:619)
```

TDA - Thread Dump Analyzer 2.0 47,6MB/95,6MB

Figure 1.3. Thread Dump View

If you want to focus on a lock found in a thread you can just click on the monitor, TDA will then expand the *Monitor* node of the thread dump and focuses the clicked monitor. It will also give you information about the monitor, if there is something special about, e.g. it has no obvious thread locking the monitor but others waiting for it.

Starting here you can easily see if a thread is hanging and holding a lock which a lot of other threads are waiting for. If there aren't any locking threads but only waiting threads it is very likely the garbage collector is locking the monitor currently. In this case you will get a hint telling you about this.



```
/home/irockel/tda/catalina.out
  Logfile
  Dump No. 1 at line 2627 around 2008-09-03 10:14:52
    Threads (162 Threads overall)
    Threads sleeping on Monitors (131 Threads sleeping)
    Threads locking Monitors (139 Threads locking)
    Monitors (125 Monitors)
  Dump No. 2 at line 4027 around 2008-09-03 10:15:03
  Dump No. 3 at line 5427 around 2008-09-03 10:16:35
```

```
<0x00007f5affc47bb0> (a tv.interactiv.j2ee.jcs.nio.JcsNioWorker):   1 Thread(s) sleeping,
<0x00007f5affc478e0> (a tv.interactiv.j2ee.jcs.nio.JcsNioWorker):   1 Thread(s) sleeping,
<0x00007f5affc47d90> (a tv.interactiv.j2ee.jcs.nio.JcsNioWorker):   1 Thread(s) sleeping,
<0x00007f5affa9b628> (a java.lang.ref.ReferenceQueue$Lock):   1 Thread(s) sleeping, 0
<0x00007f5aff98f880> (a java.util.TaskQueue):   1 Thread(s) sleeping, 0 Thread(s) waiti
<0x00007f5affc47b60> (a tv.interactiv.j2ee.jcs.nio.JcsNioWorker):   1 Thread(s) sleeping,
<0x00007f5affc49d10> (a sun.nio.ch.Util$1):   0 Thread(s) sleeping, 0 Thread(s) waiting,
<0x00007f5affc47f48> (a tv.interactiv.j2ee.jcs.nio.JcsNioWorker):   1 Thread(s) sleeping,
<0x00007f5affc474a8> (a tv.interactiv.j2ee.jcs.nio.JcsNioWorker):   1 Thread(s) sleeping,
<0x00007f5b2a375fe8> (a java.lang.Object):   10 Thread(s) sleeping, 0 Thread(s) waitin
<0x00007f5affc47818> (a tv.interactiv.j2ee.jcs.nio.JcsNioWorker):   1 Thread(s) sleeping,
<0x00007f5affc47ef8> (a tv.interactiv.j2ee.jcs.nio.JcsNioWorker):   1 Thread(s) sleeping,
<0x00007f5affc47f20> (a tv.interactiv.j2ee.jcs.nio.JcsNioWorker):   1 Thread(s) sleeping,
```

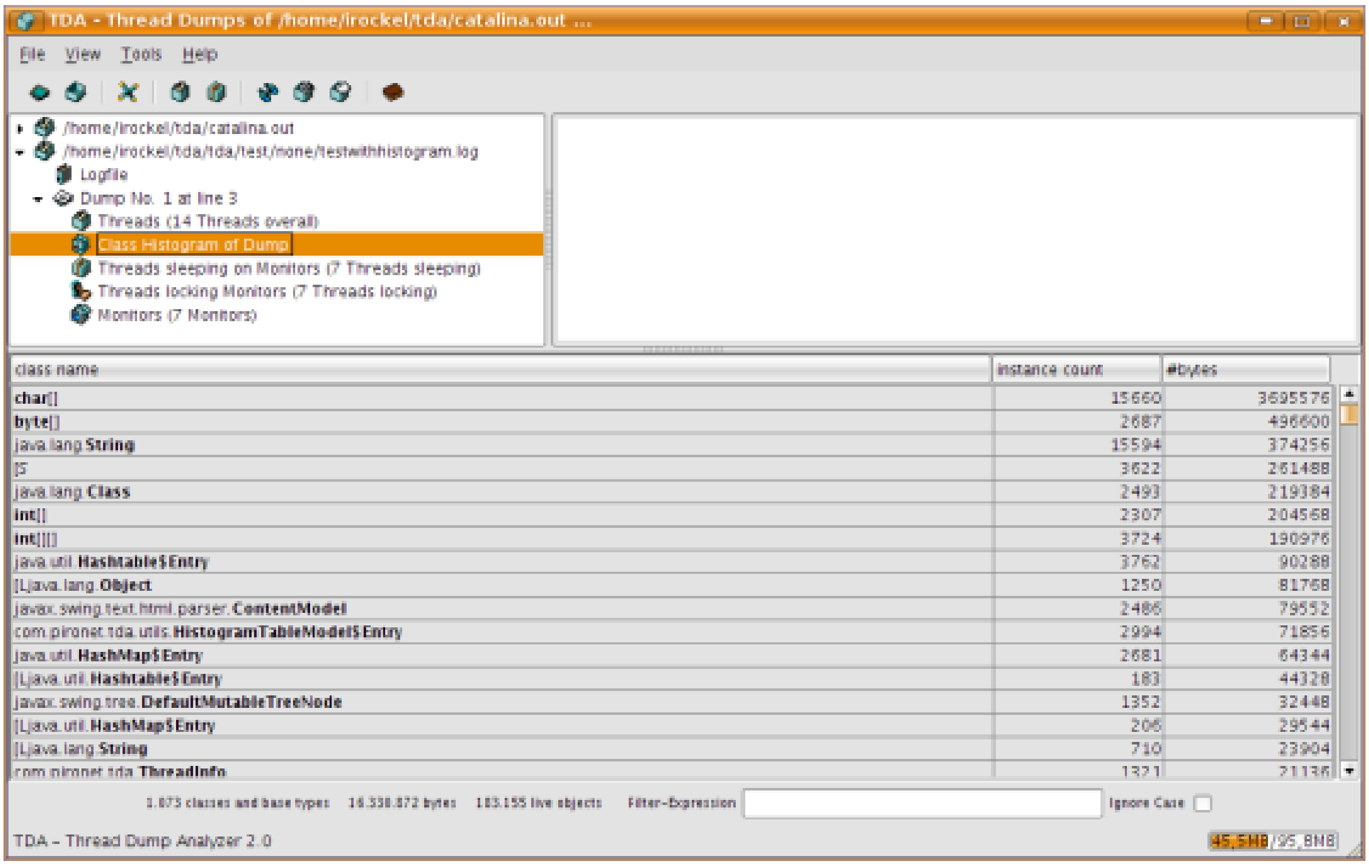
Threads locking monitor	1
Threads sleeping on monitor	1
Threads waiting to lock monitor	0

Figure 1.4. Monitors used by threads

If you added the -XX:+PrintClassHistogram to the VM-Parameters you will also see the class histogram for a thread dump, presented as node of the dump. Here you can examine all objects in the heap at the time of the thread dump. You can sort this view and filter it using the *Filter Expression*.

IMPORTANT

There is a bug in early 1.5.x releases preventing the class histogram to be printed if any other than the concurrent mark sweep garbage collector is used.


TDA - Thread Dumps of /home/irockel/tda/catalina.out ...
File View Tools Help
/home/irockel/tda/catalina.out
/home/irockel/tda/tda/test/none/testwithhistogram.log
Logfile
Dump No. 1 at line 3
Threads (14 Threads overall)
Class Histogram of Dump
Threads sleeping on Monitors (7 Threads sleeping)
Threads locking Monitors (7 Threads locking)
Monitors (7 Monitors)

class name	instance count	#bytes
char[]	15660	3695576
byte[]	2687	496600
java.lang.String	15594	374256
[S	3622	261488
java.lang.Class	2493	219384
int[]	2307	204568
int[][]	3724	190976
java.util.Hashtable$Entry	3762	90288
[Ljava.lang.Object	1250	81768
javax.swing.text.html.parser.ContentModel	2486	79552
com.pironet.tda.utils.HistogramTableModel$Entry	2994	71856
java.util.HashMap$Entry	2681	64344
[Ljava.util.Hashtable$Entry	183	44328
javax.swing.tree.DefaultMutableTreeNode	1352	32448
[Ljava.util.HashMap$Entry	206	29544
[Ljava.lang.String	710	23904
com.pironet.tda.ThreadInfo	1321	21136

1.073 classes and base types 16.338.872 bytes 183.155 live objects Filter-Expression Ignore Case ☐

TDA - Thread Dump Analyzer 2.0 45,5MB/95,8MB

Figure 1.5. Class Histogram of selected dump

If you use the loggc option with your VM to log the garbage collection information into a different log file, the class histogram will go into this log file instead of standard out and the TDA will not find it by default. To add the class histogram from a log-gc file you need to use *Open loggc file...* in the popup menu of the thread dump pane. TDA will then parse this file backward and adds the found class histograms to the thread dumps starting with the last dump. You can added multiple loggc-files, TDA will then continue after the last one where it added a class histogram with the last loggc-logfile or uses the dump you clicked on.

If your log file contains a lot of thread dumps done during one session of the VM, you can use *Find long running threads* in the tools menu to extract long running threads from the thread dumps. Currently this does also show waiting threads, so you need to search for threads actually doing something. Use appropriate filters to filter out all idle and uninteresting threads from the result.

1.3. Regular Expression For Time Stamps

Because the Sun JDK (<=1.5.x) doesn't provide any time information when dumping threads, a regular expression time stamp parsing for each line in the log file is included. TDA matches every line to the regular expression given in the preferences and stores the first matching group if the regular expression matches the line. It then takes the last match found as time stamp for the next thread dump to get a time stamp nearest to the thread dump. An example for such a regular expression is

```
(\d\d\/\d\d\/\d\d\s\d\d:\d\d:\d\d).*
```

This expression matches lines like

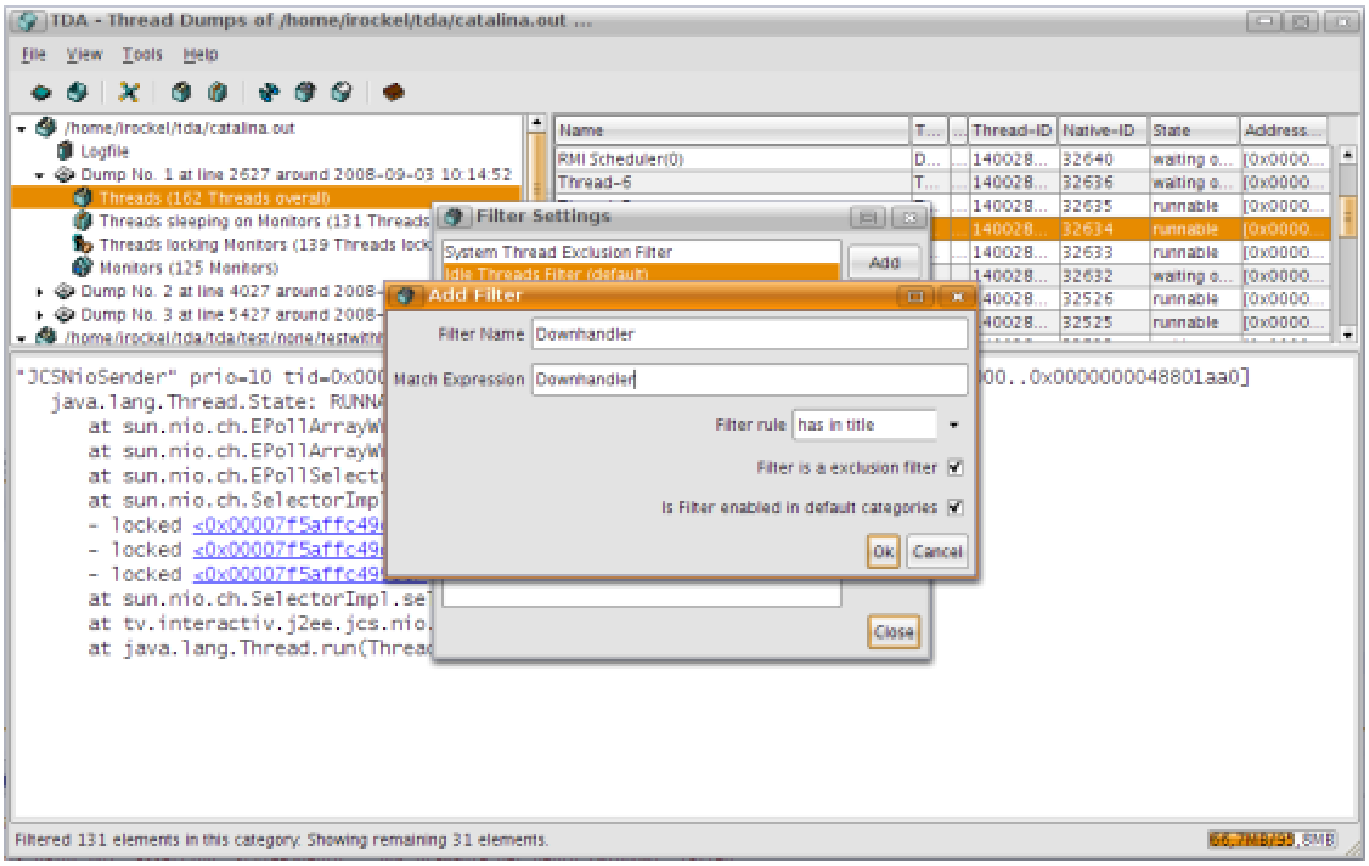
```
06/02/14 14:54:04       at java.lang.Thread.run(Thread.java:534)
```

TDA takes the first capturing group of the expression as time stamp. In this example the \d is for digits and the capturing group is included in the brackets and matches
<i>06/02/14 14:54:04</i>
. This is stored as timestamp for the next thread dump. Starting with JDK 1.6 the SUN JDKs print out a time stamp in the line before the dump. There is a default regular expression which recognizes this time stamp.

If the timestamp is stored in milliseconds since 1970 there is a checkbox in the preferences *Parsed timestamp is a long representing msecs since 1970* to tell TDA to convert the parsed time stamp from milliseconds into a human readable time stamp. See References for detailed information about the regular expressions in java.

1.4. Using Filters

To filter out sleeping and uninteresting threads in your view you can use filters to just ignore these threads. For example if you use jgroups in your application and don't want to see als the STACKs and QUEUEs of jgroups you can specify something like that seen in the next screenshot.



/home/irockel/tda/catalina.out
Logfile
Dump No. 1 at line 2627 around 2008-09-03 10:14:52
Threads (162 Threads overall)
Threads sleeping on Monitors (131 Threads
Threads locking Monitors (139 Threads lock
Monitors (125 Monitors)
Dump No. 2 at line 4027 around 2008-
Dump No. 3 at line 5427 around 2008-
/home/irockel/tda/tda/test/none/testwith

Name	T...	...	Thread-ID	Native-ID	State	Address...
RMI Scheduler(0)	D...	...	140028...	32640	waiting o...	[0x0000...
Thread-6	T...	...	140028...	32636	waiting o...	[0x0000...
		...	140028...	32635	runnable	[0x0000...
		...	140028...	32634	runnable	[0x0000...
		...	140028...	32633	runnable	[0x0000...
		...	140028...	32632	waiting o...	[0x0000...
			140028...	32526	runnable	[0x0000...
			140028...	32525	runnable	[0x0000...

```
"JCSNioSender" prio=10 tid=0x000                                     00..0x0000000048801aa0]
   java.lang.Thread.State: RUNNA
        at sun.nio.ch.EPollArrayW
        at sun.nio.ch.EPollArrayW
        at sun.nio.ch.EPollSelect
        at sun.nio.ch.SelectorImp
        - locked <0x00007f5affc49
        - locked <0x00007f5affc49
        - locked <0x00007f5affc49
        at sun.nio.ch.SelectorImpl.se
        at tv.interactiv.j2ee.jcs.nio.
        at java.lang.Thread.run(Threa
```

System Thread Exclusion Filter
Idle Threads Filter (default)

Filter Name: Downhandler
Match Expression: Downhandler
Filter rule: has in title
Filter is a exclusion filter ✓
Is Filter enabled in default categories ✓

Filtered 131 elements in this category. Showing remaining 31 elements.
66,7MB/95,5MB

Figure 1.6. A filter filtering jgroups threads

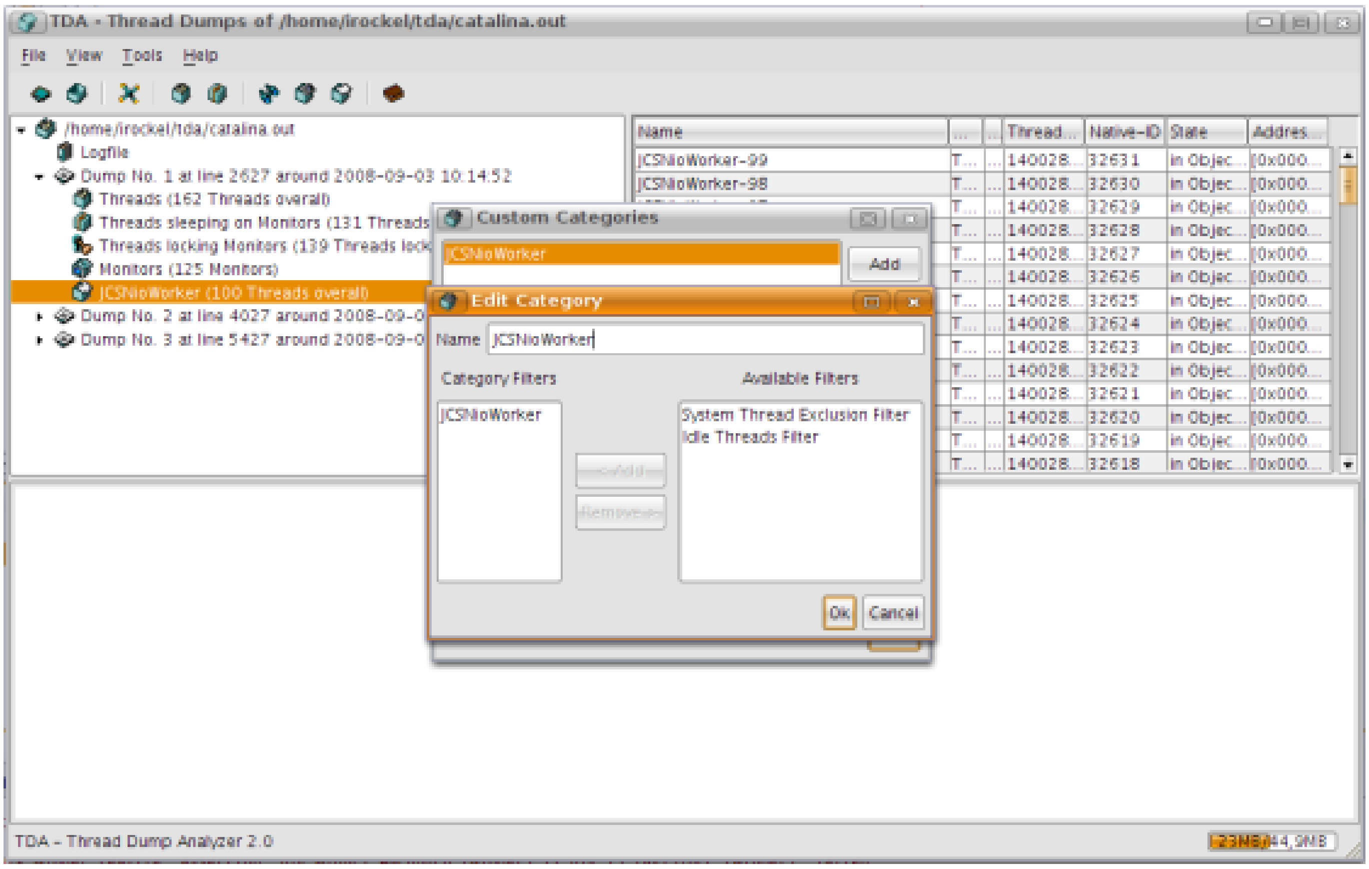
The status bar will provide information about the number of filtered threads and how many are remaining. Note that changes to the filters only apply after you changed a view. The filter changes will not be applied to the currently displayed thread view.

Filters can be used to match different parts of the thread information. You can match or search in stack, title and you can check for the thread state the thread is in. Filters can be including or excluding. The example above is a excluding filter which checks if org.jgroups is in the stack information of a thread.

If a specified filter should be applied to all displayed categories, the Default checkbox needs to be checked. Filters which aren't applied to all categories can be used in custom categories.

1.5. Using Categories

If you have special groups of threads in your application or application environment, e.g. request pool threads, you can define custom thread dump categories for grouping these threads so you can easier access them if you want to check for specific data in these threads.



TDA - Thread Dumps of /home/irockel/tda/catalina.out

/home/irockel/tda/catalina.out
- Logfile
- Dump No. 1 at line 2627 around 2008-09-03 10:14:52
  - Threads (162 Threads overall)
  - Threads sleeping on Monitors (131 Threads
  - Threads locking Monitors (139 Threads lock
  - Monitors (125 Monitors)
  - JCSNioWorker (100 Threads overall)
- Dump No. 2 at line 4027 around 2008-09-0
- Dump No. 3 at line 5427 around 2008-09-0

Name	...	...	Thread...	Native-ID	State	Addres...
JCSNioWorker-99	T...	...	140028...	32631	in Objec...	[0x000...
JCSNioWorker-98	T...	...	140028...	32630	in Objec...	[0x000...
	T...	...	140028...	32629	in Objec...	[0x000...
	T...	...	140028...	32628	in Objec...	[0x000...
	T...	...	140028...	32627	in Objec...	[0x000...
	T...	...	140028...	32626	in Objec...	[0x000...
	T...	...	140028...	32625	in Objec...	[0x000...
	T...	...	140028...	32624	in Objec...	[0x000...
	T...	...	140028...	32623	in Objec...	[0x000...
	T...	...	140028...	32622	in Objec...	[0x000...
	T...	...	140028...	32621	in Objec...	[0x000...
	T...	...	140028...	32620	in Objec...	[0x000...
	T...	...	140028...	32619	in Objec...	[0x000...
	T...	...	140028...	32618	in Objec...	[0x000...

Custom Categories

JCSNioWorker

Edit Category

Name: JCSNioWorker

Category Filters: JCSNioWorker

Available Filters: System Thread Exclusion Filter; Idle Threads Filter

TDA - Thread Dump Analyzer 2.0 — 23MB/44,9MB

Figure 1.7. A Custom Category

The Screenshot shows a custom category grouping all pool threads in an application using Java NIO as API to parse input stream data. A category consist of at least one filter. For this category a filter *JCSNioFilter* was defined which filters all these threads and which is inactive as default filter so it isn't applied to the other categories (default and custom ones). If a category doesn't contain any threads it is automatically hidden.

1.6. Deadlocks

If the JVM finds a java level deadlock in your application it logs this into the thread dump. This information will be displayed in the deadlock node of the dump. There is also a hint concerning the found deadlock(s) in the dump summary. Note though, it doesn't have any deadlock detection for java level deadlocks itself, it relies on the deadlock detection of the VM.

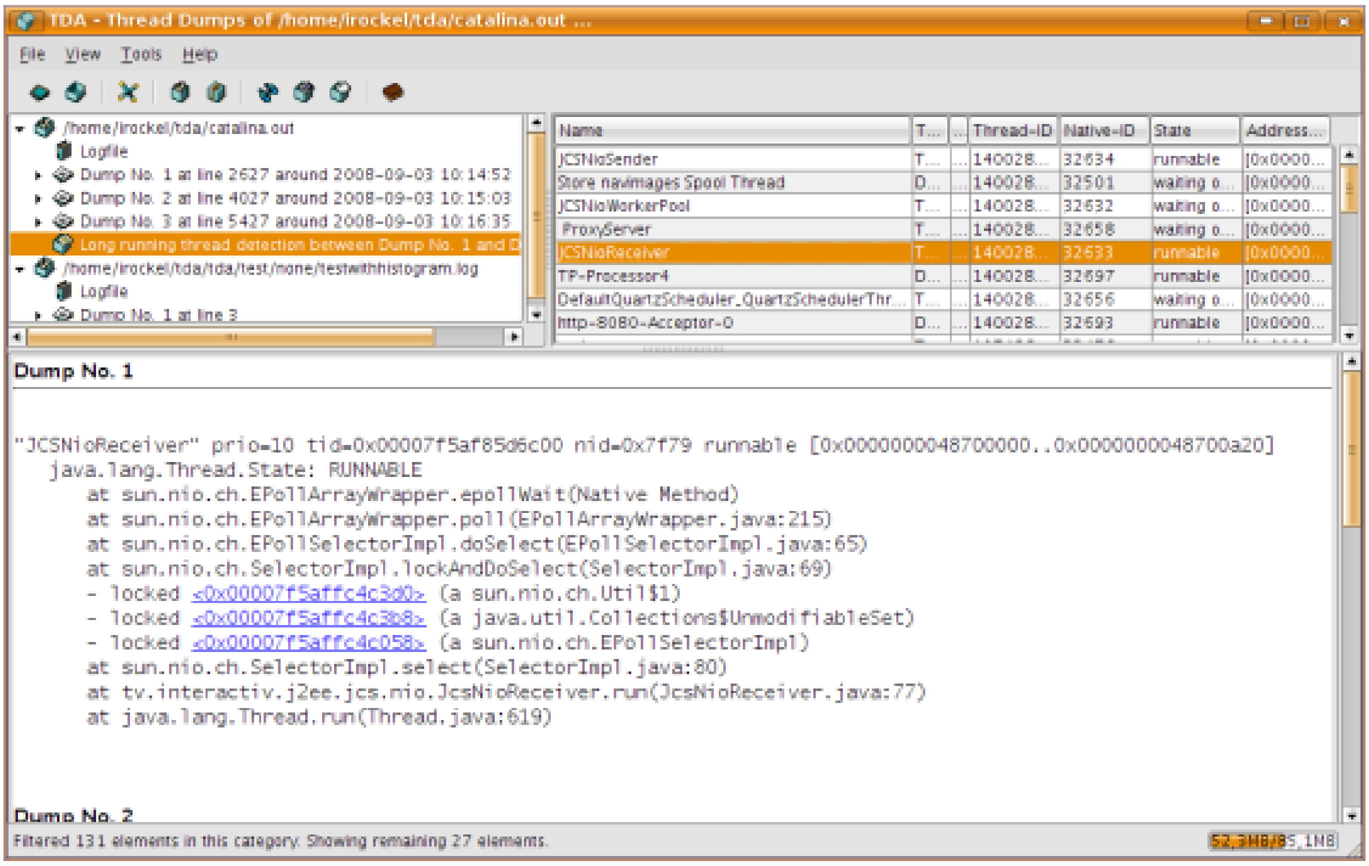
TDA also tries to find deadlocks which might be caused by some external resources or some remote communication and can't be found by the Java VM. If there are indication for such a deadlock, TDA will display information concerning this in the dump summury and gives you hints what to do next.

1.7. Sessions

If you opened several logfiles and you want to quit TDA and go back to this logfiles later, you can store the open logfiles to a session file. The logfile tree then is dumped into a session file for later usage. You must keep the logfiles if you want to be able to browse them again. For the tree navigation they are not necessary. Note that session are only supported for one version of TDA, they are not exchangeable between different versions.

1.8. Long Running Threads

A common usecase for thread dump analysis is the long running thread detection. TDA offers some help for finding these long running threads. The "Find long running threads" option in the tools menu or from the popup menu in the main tree checks for long running threads in marked thread dumps in the main tree.


TDA - Thread Dumps of /home/irockel/tda/catalina.out ...
File
View
Tools
Help
/home/irockel/tda/catalina.out
Logfile
Dump No. 1 at line 2627 around 2008-09-03 10:14:52
Dump No. 2 at line 4027 around 2008-09-03 10:15:03
Dump No. 3 at line 5427 around 2008-09-03 10:16:35
Long running thread detection between Dump No. 1 and D
/home/irockel/tda/tda/test/none/testwithhistogram.log
Logfile
Dump No. 1 at line 3

Name	T...	...	Thread-ID	Native-ID	State	Address...
JCSNioSender	T...	...	140028...	32634	runnable	[0x0000...
Store navimages Spool Thread	D...	...	140028...	32501	waiting o...	[0x0000...
JCSNioWorkerPool	T...	...	140028...	32632	waiting o...	[0x0000...
ProxyServer	T...	...	140028...	32658	waiting o...	[0x0000...
JCSNioReceiver	T...	...	140028...	32633	runnable	[0x0000...
TP-Processor4	D...	...	140028...	32697	runnable	[0x0000...
DefaultQuartzScheduler_QuartzSchedulerThr...	T...	...	140028...	32656	waiting o...	[0x0000...
http-8080-Acceptor-0	D...	...	140028...	32693	runnable	[0x0000...

Dump No. 1

```
"JCSNioReceiver" prio=10 tid=0x00007f5af85d6c00 nid=0x7f79 runnable [0x0000000048700000..0x0000000048700a20]
   java.lang.Thread.State: RUNNABLE
      at sun.nio.ch.EPollArrayWrapper.epollWait(Native Method)
      at sun.nio.ch.EPollArrayWrapper.poll(EPollArrayWrapper.java:215)
      at sun.nio.ch.EPollSelectorImpl.doSelect(EPollSelectorImpl.java:65)
      at sun.nio.ch.SelectorImpl.lockAndDoSelect(SelectorImpl.java:69)
      - locked <0x00007f5affc4c3d0> (a sun.nio.ch.Util$1)
      - locked <0x00007f5affc4c3b8> (a java.util.Collections$UnmodifiableSet)
      - locked <0x00007f5affc4c058> (a sun.nio.ch.EPollSelectorImpl)
      at sun.nio.ch.SelectorImpl.select(SelectorImpl.java:80)
      at tv.interactiv.j2ee.jcs.nio.JcsNioReceiver.run(JcsNioReceiver.java:77)
      at java.lang.Thread.run(Thread.java:619)
```

Dump No. 2

Filtered 131 elements in this category. Showing remaining 27 elements.

52,3MB/85,1MB

Figure 1.8. Long running threads detection

The analysis is quite simple though, it just compares the thread names and if they match adds the thread to the result. A regular expression can be specified for filtering just the interesting threads. The usual filters are also applied to the result view of the analysis to filter out uninteresting threads. For a useful result appropriate filters should be specified to filter out all sleeping and uninteresting threads.

HINT

The long running threads detection is a bit picky about the selected nodes for the detection. You should only select thread dump root nodes otherwise the detection might through an exception as it doesn't know what to do with the selected nodes. You also always should select nodes from one VM run, otherwise you might get quite weird results because the detection matched threads from different VM runs.

1.9. Heavy Load Analysis

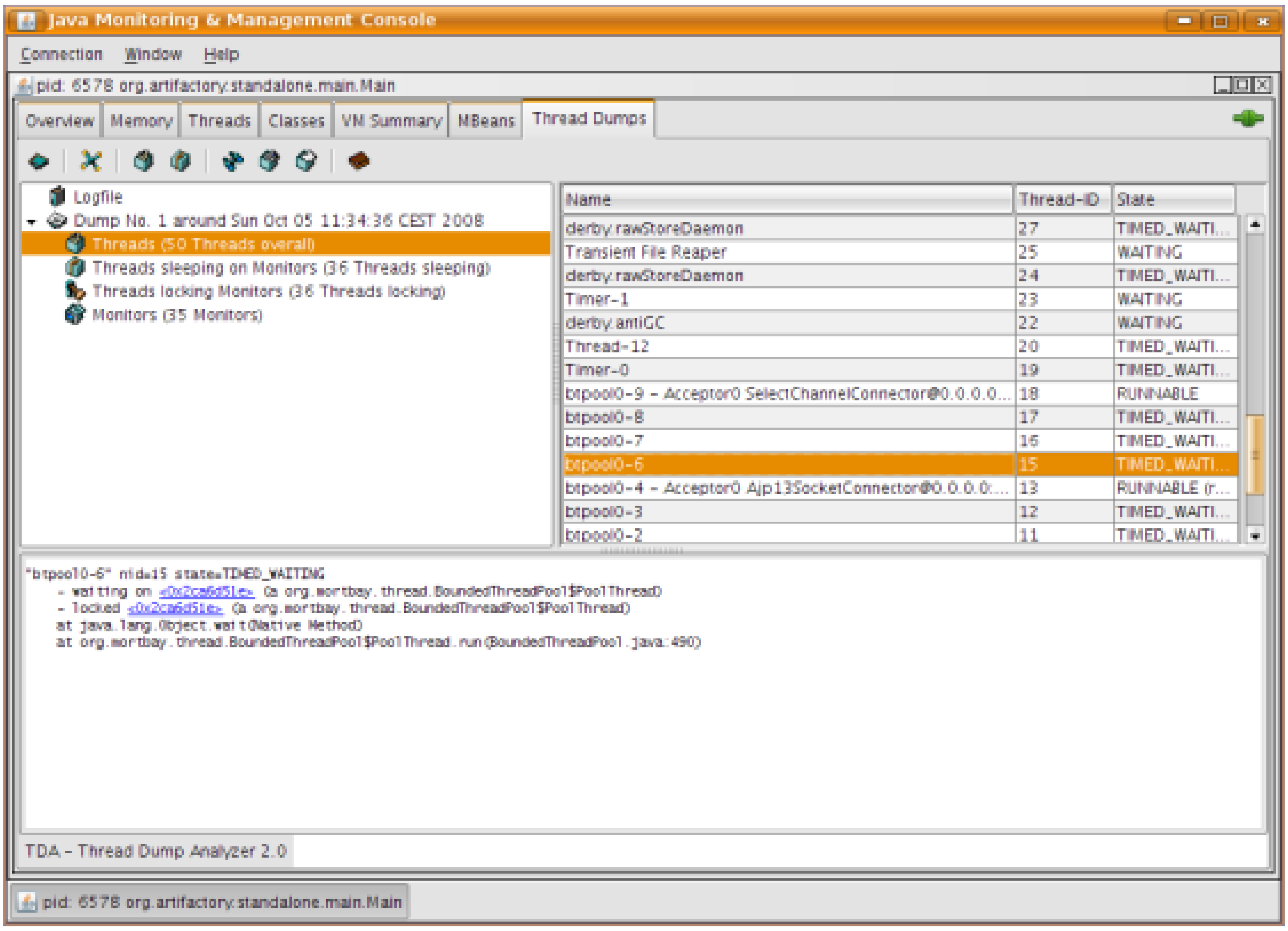
If a system is under heavy load, several thread dumps should be taken (e.g. every two minutes over a short period). The threads all have a thread id which can be mapped to real process ID on Linux/Solaris Systems. TDA parses the ID and displays it as "Native ID" in the thread view. Using top in light-weight-process view on Linux and prstat on Solaris with the process id of the java process where the thread dumps came from you can match this native ID to the threads in the dump. This way it is easy to identify threads which produce heavy load as the thread view in top and prstat shows the cpu load of each single thread.

1.10. JConsole Plugin

TDA can be used as plugin in the JVM utility JConsole. Use the following to start JConsole with TDA as plugin:

```
jconsole -pluginpath ./tda.jar
```

The javahelp jar *jhall.jar* needs to be in the same directory as the tda.jar or in the subdirectory *lib*, otherwise clicking the help button would lead to an exception. The Plugin offers requesting thread dumps from a remote Virtual Machine through JMX. The Dumps areparsed and can be analyzed just like running as standalone-application. The upper-right-frame showing the dump tree has a popup menu offering the plugin features.


Java Monitoring & Management Console
Connection Window Help
pid: 6578 org.artifactory.standalone.main.Main
Overview Memory Threads Classes VM Summary MBeans Thread Dumps
Logfile
Dump No. 1 around Sun Oct 05 11:34:36 CEST 2008
Threads (50 Threads overall)
Threads sleeping on Monitors (36 Threads sleeping)
Threads locking Monitors (36 Threads locking)
Monitors (35 Monitors)
TDA - Thread Dump Analyzer 2.0
pid: 6578 org.artifactory.standalone.main.Main

Name	Thread-ID	State
derby.rawStoreDaemon	27	TIMED_WAITI...
Transient File Reaper	25	WAITING
derby.rawStoreDaemon	24	TIMED_WAITI...
Timer-1	23	WAITING
derby.antiGC	22	WAITING
Thread-12	20	TIMED_WAITI...
Timer-0	19	TIMED_WAITI...
btpool0-9 - Acceptor0 SelectChannelConnector@0.0.0.0...	18	RUNNABLE
btpool0-8	17	TIMED_WAITI...
btpool0-7	16	TIMED_WAITI...
btpool0-6	15	TIMED_WAITI...
btpool0-4 - Acceptor0 Ajp13SocketConnector@0.0.0.0:...	13	RUNNABLE (r...
btpool0-3	12	TIMED_WAITI...
btpool0-2	11	TIMED_WAITI...

```
"btpool0-6" nid=15 state=TIMED_WAITING
    - waiting on <0x2ca6d51e> (a org.mortbay.thread.BoundedThreadPool$PoolThread)
    - locked <0x2ca6d51e> (a org.mortbay.thread.BoundedThreadPool$PoolThread)
    at java.lang.Object.wait(Native Method)
    at org.mortbay.thread.BoundedThreadPool$PoolThread.run(BoundedThreadPool.java:490)
```

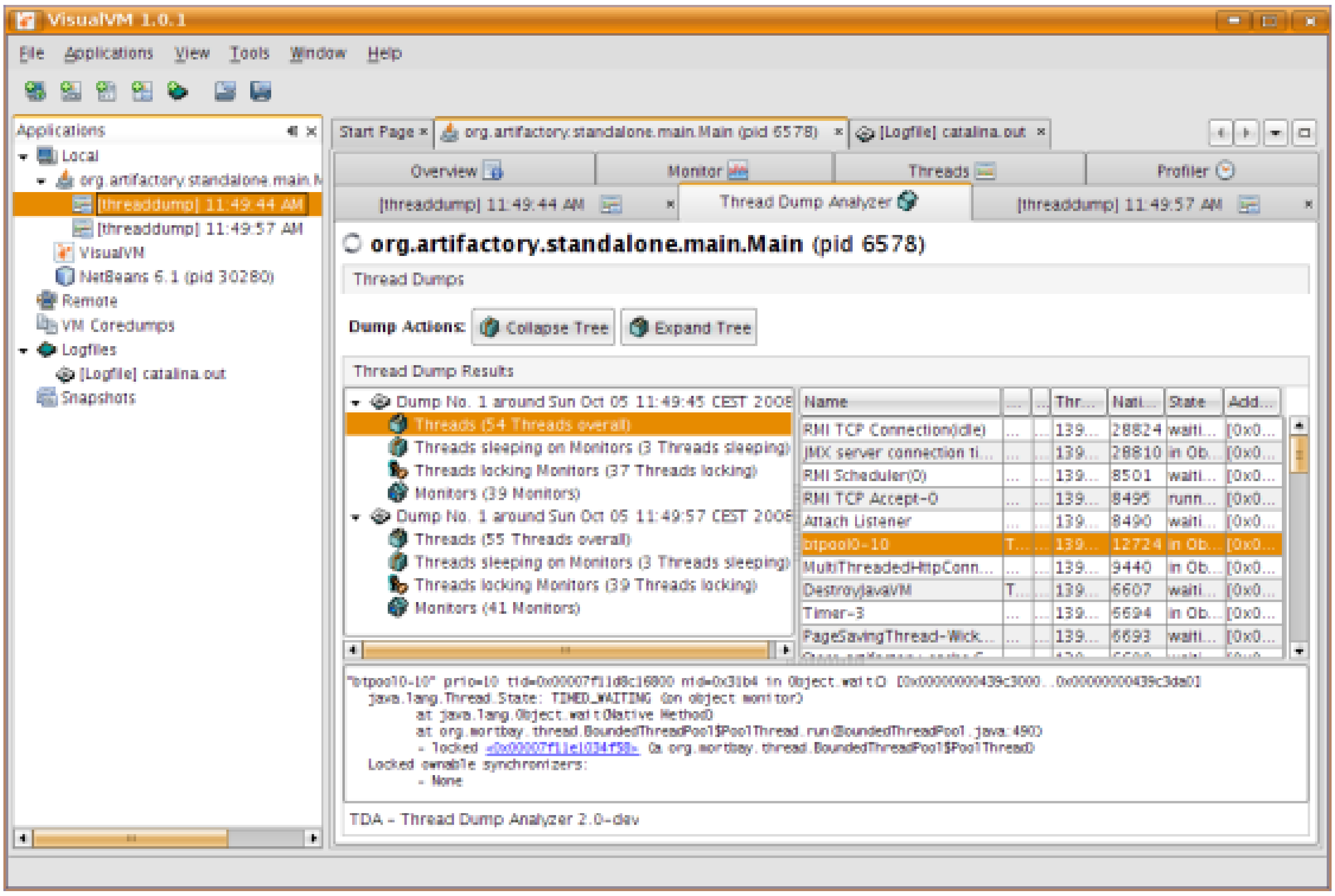
Figure 1.9. TDA JConsole Plugin

The toolbar is also available for convenience, it can be switched off using the popup menu. The requested Thread Dumps can also be stored as Logfile for later offline analysis. The plugin supports requesting thread dumps from 1.5 and 1.6 VMs. The extended informations from 1.6 are not parsed yet though (e.g. informations about locked synchronizers). But they are saved to the logfile also.

1.11. VisualVM Plugin

TDA can also be included into VisualVM[1]. It is available as plugin from the VisualVM Plugin Center. Note though, if you use jvisualvm from recent Sun JDK's you wont find it in the plugin center but you can still download it manually from the TDA Website from the Document and Files section. All three files available for download need to be installed.

[1] https://visualvm.dev.java.net


VisualVM 1.0.1
File Applications View Tools Window Help
Applications
Local
org.artifactory.standalone.main.M
[threaddump] 11:49:44 AM
[threaddump] 11:49:57 AM
VisualVM
NetBeans 6.1 (pid 30280)
Remote
VM Coredumps
Logfiles
[Logfile] catalina.out
Snapshots
Start Page
org.artifactory.standalone.main.Main (pid 6578)
[Logfile] catalina.out
Overview
Monitor
Threads
Profiler
[threaddump] 11:49:44 AM
Thread Dump Analyzer
[threaddump] 11:49:57 AM
org.artifactory.standalone.main.Main (pid 6578)
Thread Dumps
Dump Actions:
Collapse Tree
Expand Tree
Thread Dump Results
Dump No. 1 around Sun Oct 05 11:49:45 CEST 2008
Threads (54 Threads overall)
Threads sleeping on Monitors (3 Threads sleeping)
Threads locking Monitors (37 Threads locking)
Monitors (39 Monitors)
Dump No. 1 around Sun Oct 05 11:49:57 CEST 2008
Threads (55 Threads overall)
Threads sleeping on Monitors (3 Threads sleeping)
Threads locking Monitors (39 Threads locking)
Monitors (41 Monitors)
TDA - Thread Dump Analyzer 2.0-dev

Name	...	...	Thr...	Nati...	State	Add...
RMI TCP Connection(idle)	...	...	139...	28824	waiti...	[0x0...
JMX server connection ti...	...	...	139...	28810	in Ob...	[0x0...
RMI Scheduler(0)	...	...	139...	8501	waiti...	[0x0...
RMI TCP Accept-0	...	...	139...	8495	runn...	[0x0...
Attach Listener	...	...	139...	8490	waiti...	[0x0...
btpool0-10	T...	...	139...	12724	in Ob...	[0x0...
MultiThreadedHttpConn...	...	...	139...	9440	in Ob...	[0x0...
DestroyJavaVM	T...	...	139...	6607	waiti...	[0x0...
Timer-3	...	...	139...	6694	in Ob...	[0x0...
PageSavingThread-Wick...	...	...	139...	6693	waiti...	[0x0...

```
"btpool0-10" prio=10 tid=0x00007f11d8c16800 nid=0x31b4 in Object.wait() [0x00000000439c3000..0x00000000439c3da0]
   java.lang.Thread.State: TIMED_WAITING (on object monitor)
        at java.lang.Object.wait(Native Method)
        at org.mortbay.thread.BoundedThreadPool$PoolThread.run(BoundedThreadPool.java:490)
        - locked <0x00007f11e1034f58> (a org.mortbay.thread.BoundedThreadPool$PoolThread)
   Locked ownable synchronizers:
        - None
```

Figure 1.10. TDA VisualVM Plugin

The Plugin attaches itself to the thread dump feature in VVM and offers a TDA View of the application thread dumps and the navigation tree offers a *Logfile* section, where you can add logfiles containing thread dumps.

The TDA help is only available as a limited subset of the full help because VisualVM doesn't include JavaHelp. To get the full help, either download the documentation from the TDA Website or start TDA stand-alone.

2 Analyzing production environments

Starting with Sun's JDK 1.5 it is quite easy to get information about a badly behaving application in a production environment using the JVM tools jmap and jstack. Jmap fetches a heap dump from the VM the application is running in and jstack fetches the thread dumps. These information can be taken from the server the application server is running on and analyzed offline. Even without remote access to the server you still can get these dumps from your customer. Many Monitoring tools usually need some kind of remote access.

Imagine a customer's web application with a lot of concurrent users accessing the system and the system is under heavy load. The system is only accessed during office hours. But you have noticed the load of the system stays high in the night although nobody is accessing it. What can you do?

2.1. Dumping Information about running threads

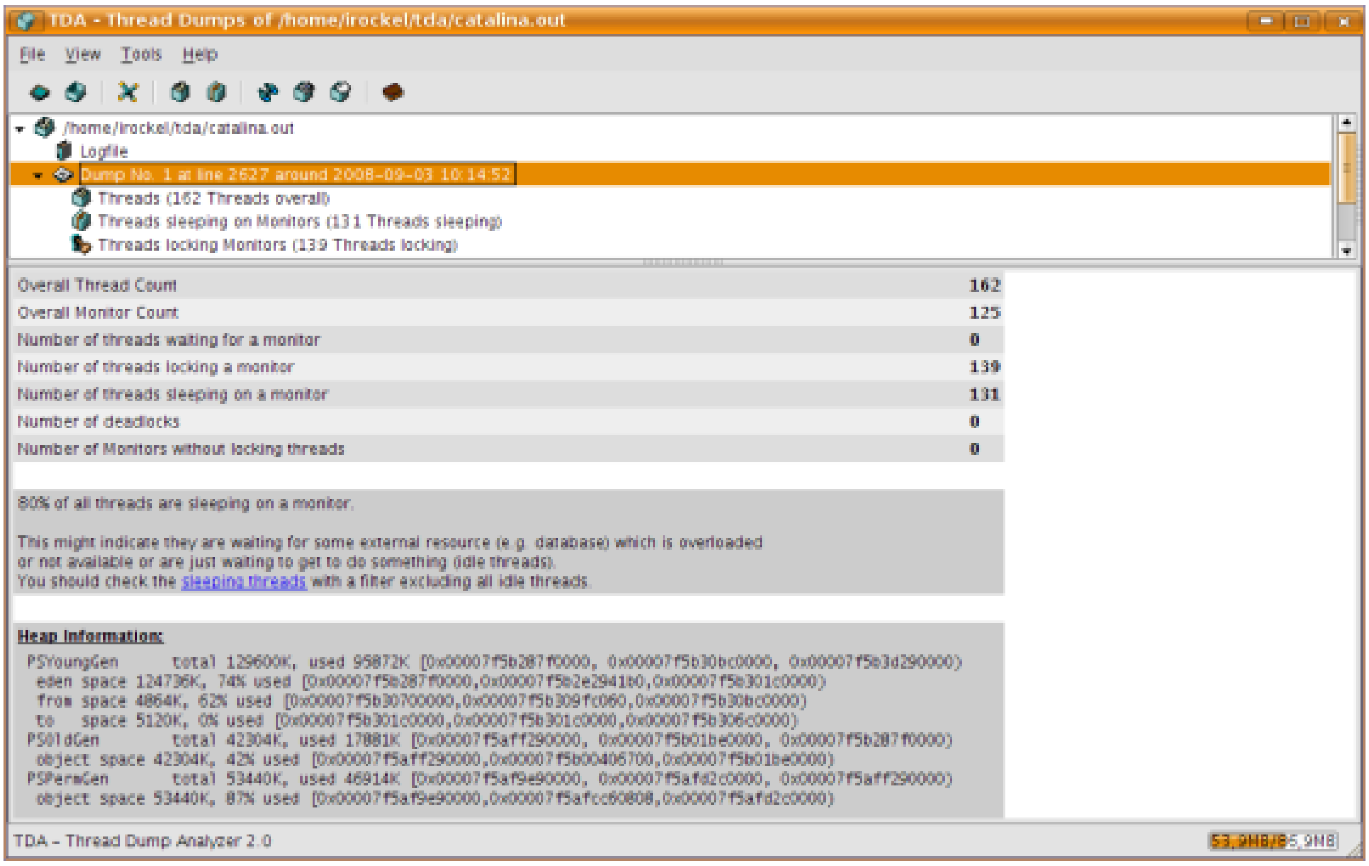
First of all you can fetch several thread dumps either using jstack or using kill -QUIT if you're using an older VM than 1.5. Using these dumps you can easily detect long running (looping?) threads which might cause the load. You should fetch about five dumps and wait several seconds between fetching, lets say 20 seconds. The command line with jstack then looks something like this

```
jstack <pid> > dump.log
```

for the first dump and

```
jstack  >> dump.log
```

for the additional dumps. You then can use the TDA - Thread Dump Analyzer to analyze the thread dumps to get an idea what is currently happening in your application. TDA will try to give you some hints about what might be wrong in the dumps (e.g. a lot of threads are waiting for the garbage collector) like in the screenshot below.


TDA - Thread Dumps of /home/irockel/tda/catalina.out
File View Tools Help
/home/irockel/tda/catalina.out
Logfile
Dump No. 1 at line 2627 around 2008-09-03 10:14:52
Threads (162 Threads overall)
Threads sleeping on Monitors (131 Threads sleeping)
Threads locking Monitors (139 Threads locking)
53,9MB/85,9MB

Overall Thread Count	**162**
Overall Monitor Count	**125**
Number of threads waiting for a monitor	**0**
Number of threads locking a monitor	**139**
Number of threads sleeping on a monitor	**131**
Number of deadlocks	**0**
Number of Monitors without locking threads	**0**

80% of all threads are sleeping on a monitor.

This might indicate they are waiting for some external resource (e.g. database) which is overloaded
or not available or are just waiting to get to do something (idle threads).
You should check the sleeping threads with a filter excluding all idle threads.

Heap Information:

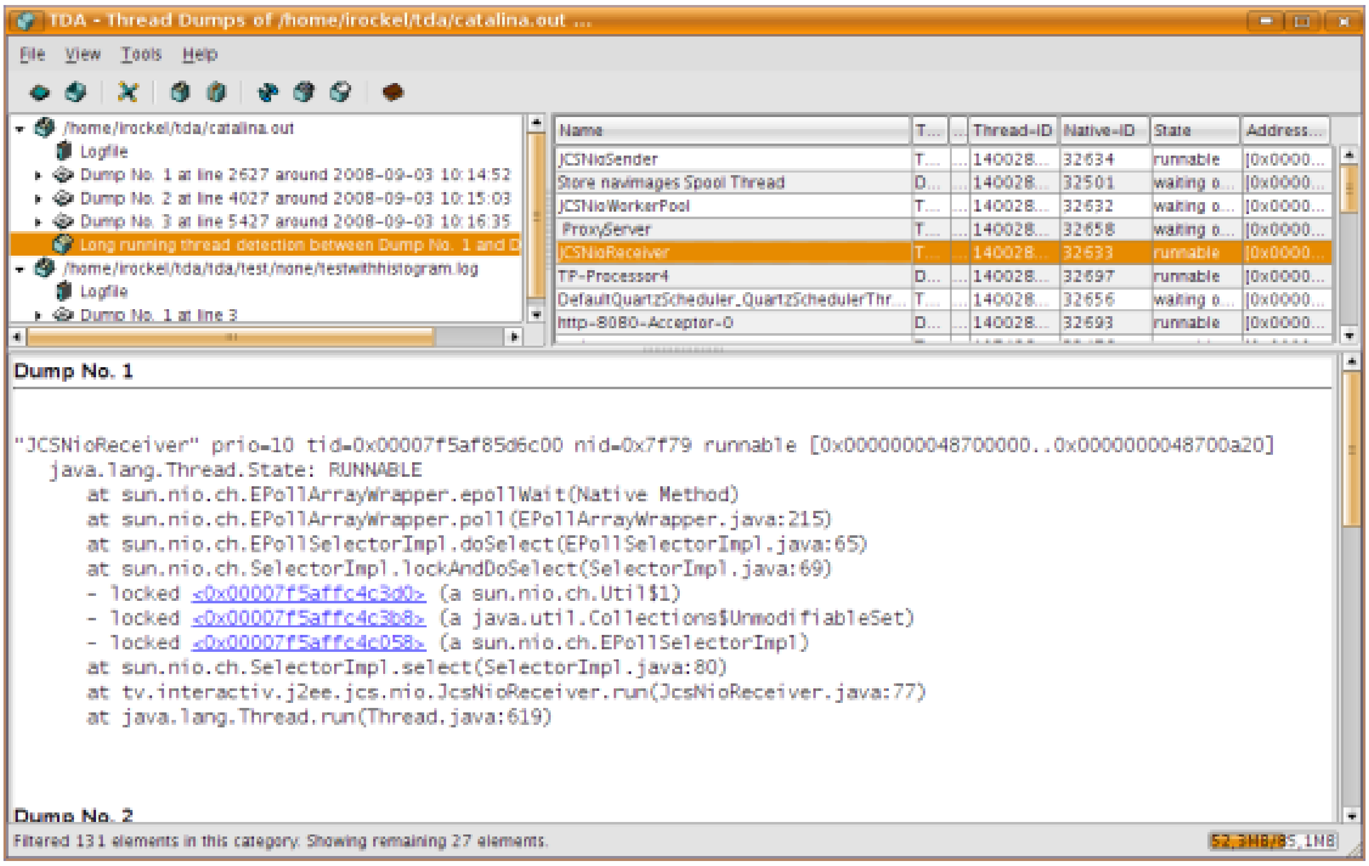
```
 PSYoungGen      total 129600K, used 95872K [0x00007f5b287f0000, 0x00007f5b30bc0000, 0x00007f5b3d290000)
  eden space 124736K, 74% used [0x00007f5b287f0000,0x00007f5b2e294180,0x00007f5b301c0000)
  from space 4864K, 62% used [0x00007f5b30700000,0x00007f5b309fc060,0x00007f5b30bc0000)
  to   space 5120K, 0% used [0x00007f5b301c0000,0x00007f5b301c0000,0x00007f5b306c0000)
 PSOldGen        total 42304K, used 17881K [0x00007f5aff290000, 0x00007f5b01be0000, 0x00007f5b287f0000)
  object space 42304K, 42% used [0x00007f5aff290000,0x00007f5b00406700,0x00007f5b01be0000)
 PSPermGen       total 53440K, used 46914K [0x00007f5af9e90000, 0x00007f5afd2c0000, 0x00007f5aff290000)
  object space 53440K, 87% used [0x00007f5af9e90000,0x00007f5afcc60808,0x00007f5afd2c0000)
```

TDA - Thread Dump Analyzer 2.0

Figure 2.1. TDA giving hints about selected dump

High load without anybody accessing the application usually either means the garbage collector is running endlessly because the application is very low on memory and the garbage collector is unable to free enough memory (TDA will give you a hint on this) or there are some threads (or just one) looping and running endlessly. TDA tries to filter out all idle threads to make it easier for you to find really running threads which are either running or are waiting for some external resources (it will give you a hint regarding the last issue). You need to enable these filters in the filter settings. You might also have a look at the TDA help for additional information.

For letting TDA search for long running threads mark the threads dumps you want to analyze and choose the "Find long running" threads option. TDA will then search for threads it finds in the dumps at least n-times, whereas you define n in the setting dialog for the detection. It will then present the result added to the dump tree. You should enable the idle-threads filter to filter out unimportant threads. In the screenshot below you can see a long running thread which is running in an endless loop.



TDA - Thread Dumps of /home/irockel/tda/catalina.out ...

```
/home/irockel/tda/catalina.out
    Logfile
    Dump No. 1 at line 2627 around 2008-09-03 10:14:52
    Dump No. 2 at line 4027 around 2008-09-03 10:15:03
    Dump No. 3 at line 5427 around 2008-09-03 10:16:35
    Long running thread detection between Dump No. 1 and D
/home/irockel/tda/tda/test/none/testwithhistogram.log
    Logfile
    Dump No. 1 at line 3
```

Name	T...	...	Thread-ID	Native-ID	State	Address...
JCSNioSender	T...	...	140028...	32634	runnable	[0x0000...
Store navimages Spool Thread	D...	...	140028...	32501	waiting o...	[0x0000...
JCSNioWorkerPool	T...	...	140028...	32632	waiting o...	[0x0000...
ProxyServer	T...	...	140028...	32658	waiting o...	[0x0000...
JCSNioReceiver	T...	...	140028...	32633	runnable	[0x0000...
TP-Processor4	D...	...	140028...	32697	runnable	[0x0000...
DefaultQuartzScheduler_QuartzSchedulerThr...	T...	...	140028...	32656	waiting o...	[0x0000...
http-8080-Acceptor-0	D...	...	140028...	32693	runnable	[0x0000...

Dump No. 1

```
"JCSNioReceiver" prio=10 tid=0x00007f5af85d6c00 nid=0x7f79 runnable [0x0000000048700000..0x0000000048700a20]
   java.lang.Thread.State: RUNNABLE
      at sun.nio.ch.EPollArrayWrapper.epollWait(Native Method)
      at sun.nio.ch.EPollArrayWrapper.poll(EPollArrayWrapper.java:215)
      at sun.nio.ch.EPollSelectorImpl.doSelect(EPollSelectorImpl.java:65)
      at sun.nio.ch.SelectorImpl.lockAndDoSelect(SelectorImpl.java:69)
      - locked <0x00007f5affc4c3d0> (a sun.nio.ch.Util$1)
      - locked <0x00007f5affc4c3b8> (a java.util.Collections$UnmodifiableSet)
      - locked <0x00007f5affc4c058> (a sun.nio.ch.EPollSelectorImpl)
      at sun.nio.ch.SelectorImpl.select(SelectorImpl.java:80)
      at tv.interactiv.j2ee.jcs.nio.JcsNioReceiver.run(JcsNioReceiver.java:77)
      at java.lang.Thread.run(Thread.java:619)
```

Dump No. 2

Filtered 131 elements in this category. Showing remaining 27 elements.

Figure 2.2. Long running threads detection

2.2. Long running thread detection on Solaris

If your application is running on Solaris and you have remote access to the machine you can even determine directly the thread which might be looping by using prstat -p <app-server-vm-pid>. prstat will show you all threads running in your application and how much CPU time each of them is consuming. The pids shown there (in decimal) can be matched to the nids (in hexadecimal) in the thread dumps.

2.3. Analyzing the heap profile

I use the YourKit Profiler (commercial but free eval-license for 15 days) to analyze the heap profiles. The tool has the quite handy "Find big objects" option which makes it very easy to find stuff which consumes a lot of memory and shouldn't be there or at least not that big. If you don't want to spend the money though, you can use the JVM tool jhat which starts a small web server, where you can connect to and analyze the heap profile.

For a small overview you can also use the class histogram feature of the TDA tool. Use either jstack to fetch a class histogram from your application VM or use the VM-option -XX:+PrintClassHistogram to have it dumped when requesting a thread dump if you are on an 1.4.x VM. Refer to the TDA help for further information.

3 Application Analysis using JConsole

With JDK 1.5 Sun introduced JConsole, a swing based tool for instrumentalizing the new JMX MBeans for VM analysis in 1.5 and to provide easy remote access to all other JMX MBeans offered (e.g. by the application server or an application). With the release of JDK 1.6 Sun enhanced the utility to also support plugins and simplified access to the applications to analyze. With JConsole it is quite easy to do some application analysis without critical impact on the application.

3.1. Remote application access

For enabling remote access to an application using JConsole you need an open port from the machine the application is running on and start the application with this port. To connect to a remote application running with JDK 1.5 you need to specify something like this:

```
-Dcom.sun.management.jmxremote.port=5555
-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false
```

I have done this for connecting to an Oracle Application Server 10.1.3.2 instance. Note: this is not the recommended setting for an production environment as is insecure. Read the jconsole documentation on how to set up a secure connection to a JVM.

3.2. Requesting Dumps

Since the 1.6 release it is also quite easy to request heap and thread dumps. These Dumps then can be parsed by tools like the SAP Memory Analyzer to analyze heap dumps and tools like Samurai or my TDA for thread dumps.

3.3. Plugins For JConsole

One interesting plugin currently available publically is an enhanced JTop Plugin (based on a demo plugin from the jdk samples). JConsole doesn't display any information about the cpu usage of the threads running in a JVM, it only displays a graph about the overall cpu usage. The JTop plugin shows enhanced information about the cpu utilization of all threads running in a JVM just like top does for processes on unix plattforms. The enhanced JTop plugin can be downloaded from Peter Doornbosch's Blog.

Appendix A. References

i. GC-Viewer[1]

ii. VisualVM[2]

iii. JConsole[3]

iv. Java API Specification 1.4.2[4] (for regular expressions look for *Pattern* class.)

v. SendSignal[5]

vi. Dumpster (my blog)[6]

[1] http://www.tagtraum.com/gcviewer.html
[2] http://visualvm.dev.java.net/
[3] http://java.sun.com/j2se/1.5.0/docs/guide/management/jconsole.html
[4] http://java.sun.com/j2se/1.4.2/docs/api/index.html
[5] http://www.latenighthacking.com/projects/2003/sendSignal/
[6] http://www.jroller.com/dumpster